REMARKS

An information disclosure statement summarizing some of the papers written by Oscar Pastor and his colleagues is enclosed along with a CD-ROM that contains copies of all those papers in PDF format.

In response to the objection to the specification regarding there being highlighted and underlined words therein, a substitute specification is enclosed herewith with the highlighting and underlining removed. No markup specification is included because to strikeout the highlighted and underlined text and then add it again underlined to indicate it has been added is non sensical. The undersigned hereby affirms that no new matter has been added to the substitute specification.

Obviousness Rejection of Claims 1-3

The Examiner has rejected claims 1-3 as obvious based upon a combination of Goodwin et al (US 6,199,195) in view of Goldberg et al. (6,571,232) and further in view of Tse (5,742,754). The Examiner is respectfully requested to withdraw this obviousness rejection based upon a combination of references on grounds there is no suggestion to make the combination.

Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention absent some teaching, suggestion or incentive to do so. In re Bond, 910 F.2d 831, 834, 15 USPQ2d 1566, 1568 (Fed. Cir. 1990). Suggestion arises from one of ordinary skill in the art perceiving a liklihood of success in solving the problem the inventors solved by making the combination. In other words, the consistent criterion for determination of obviousness is whether the prior art would have suggested to one of ordinary skill in the art that this process should be carried out *and would have a reasonable likelihood of success, viewed in the light of the prior art*. See Burlington Industries v. Quigg, 822 F.2d 1581, 1583, 3 USPQ2d 1436, 1438 (Fed.Cir.1987); In re

Hedges, 783 F.2d 1038, 1041, 228 USPQ 685, 687 (Fed.Cir.1986).

The issue then is would one skilled in the art perceive a liklihood of success in solving the problem the claimed invention solves by making a combination of the references the Examiner has combined. One aspect of this question is whether or not all the knowledge needed to make the claimed invention is found in the combined references. The answer to that questions is no becuase not all the structure and process steps and code elements which are recited in limitations of rejected claims 1-3 are found in the prior art combination applied to the claims. Therefore, even if one skilled in the art were to make the combination, the combination would fall short of the claimed invention and fail of its essential purpose. This would discourage one skilled in the art from attempting to make the combination to solve the problem the inventors solved because one skilled in the would perceive failure and not liklihood of success.

Overview of the Problem

The prosecuted application claims a process (and an apparatus and a computer program) to automatically translate a formal language specification, written in a formal language, that defines a complete computer program, into the full and complete source code of said computer program. By "full and complete" the prosecuted application claims that no third party source code, or source code from existing components or any other form of addition is needed be combined with the result of the claimed process so that said result be the computer program described by the formal language specification.

In order to do so, two conditions must hold:

- the formal language specification must enable the representation of every functional aspect of a the computer program that the claimed process produces automatically;
- the different elements that are present in the formal language specification must be combined in a certain way at execution time so that the computer program that the

claimed process produces automatically behaves as expected.

While some of the elements present in the formal language specification are found in many software development methodologies and tools (e.g: classes, attributes, relationships) some others are limitations which are not found in the prior art. This argument lists the limitations present in the prosecuted application which are not found in the prior art, neither individually nor as the result of a combination of the applied prior art reference.

As described in the prosecuted application, <u>AND AS CLAIMED IN REJECTED CLAIMS 1-3, AS AMENDED,</u> a formal language specification organizes its different elements (also referred to as *primitives*) in four different models: the Object Model, the Functional Model, the Dynamic Model and the Presentation Model. This document follows said structure to point out, the distinctive primitives, or limitations, which are not found in the prior art.

A limitation in a claim should be interpreted in accordance with the specification and prosecution history, and to do so is not reading limitations into the claim. Under a recent *in banc* decision of the Federal Circuit, claim construction is a job for the judge as it is a question of law. <u>Markman v. Westview Instruments, Inc.</u>, 52 F.2d 967 (Fed. Cir. 1995). Three intrinsic sources of data on this subject are the specification, the language of the claims themselves and the prosecution history. The judge is free to consider extrinsic sources also such as expert testimony, dictionaries, prior art references etc., but in the recent <u>Vitronics</u> decision, the Federal Circuit held that the meaning of the claims is almost always ascertainable from the intrinsic evidence and resort to extrinsic evidence is the exception rather than the rule. <u>Vitronics Corp. v. Conceptronic Inc.</u>, 39 USPQ2d 1573 (Fed. Cir. 1996). Further, <u>Vitronics</u> holds that it is legal error to rely on extrinsic evidence where the meaning of the claims is clear from the intrinsic evidence.

Accordingly, the limitations object model, dynamic model, functional model and presentation model in claims 1-3, as amended, should be interpreted in accordance with the teachings regarding these models in the specification, and to do so is not importing limitations from the specification into the claim. For each limitation, a brief description of its purpose is presented, along with the figures and/or passages in the prosecuted application where the limitation is described or referred to. A short explanation on why the prior art references do not teach the limitation follows.

Local Transactions (Object Model)

Local transactions (along with valuations in the Functional Model) are important and part of the object model limitation, AND ARE NOT FOUND IN THE PRIOR ART. These concepts are the basis for completely specifying the behaviour of a system in terms of how every object of every class of the system will react (change its state, the values of each of its attributes) in response to the occurrence of a service (local transactions or events). While valuations state how an object changes its state upon the occurrence of an event (atomic execution), transactions specify how several events combine so as to create a more complex behaviour (molecular execution).

Purpose and Support In Specification

(All references to Column and Lines numbers are references to the issued patent in the parent case of this CIP -- US 6,681,383. All references to page numbers are to the page numbers of the clean substitute specification of the continuation-in-part of the parent case included herewith)

A transaction is a molecular service: that is, a service whose functionality is defined by a sequence of other services. The services composing a local transaction can be atomic (events) or molecular (transactions).

The combination of the valuations defined in the Functional Model (which determine the functionality of events) with the composition mechanism of transactions result in the ability to completely define the functionality of services (either atomic or molecular).

Support is at page 13:

> "FIG. 13 is a screenshot of the dialog box used to create one formula in a local transaction carried out by a composed service (single services are called events, and composed services are called local transactions)." Page 13

and from page 39:

> The system classes are obtained from the object model. For each class, there are a set of constant, variable or derived attributes; a set of services, including private and shared events and local transactions; integrity constraints specified for the class; and derivation expressions corresponding to the derived attributes. For a complex class (those defined by using the provided aggregation and inheritance class operators), the object model also provides the particular characteristics specified for the corresponding complex aggregated or specialized class.

and from page 20:

> Services can be of two types: events and transactions. Events are atomic operations while transactions are composed of services which can be in turn events or transactions. Every service can have the following characteristics: name, type of service (event or transaction), service alias, remarks and a help message. Events can be of three types: new, destroy or none of them. Events can also be shared by several classes of the project. Shared events belong to all classes sharing them. Transactions have a formula that expresses the composing of services.

An "event" means a single service and not a transaction which is defined as a composed or complex service (which means more than one service executes).

Distinction over prior art

None of the prior art references teach the concept of local transactions as a means of specifying the combination of events (events being atomic execution units).

None of the references teaches either the concept of event nor the concept of valuations, thus not being obvious to someone skilled in the art that the combination of events (and their associated valuations) could lead to the specification of molecular execution units (local transactions).

If the Examiner has a passage which actually teaches an object model with local transactions, the applicants hereby request that such a passage be properly identified and requests the Examiner to explain how the passage teaches same.

Therefore, none of the references nor a combination of them could teach a process or apparatus of computer-readable medium of claims 1-3 which perform the function or drive a computer to automatically produce the complete source code of an application from a formal language specification containing an object model. This is because said source code produced by the process, apparatus or computer controlled by the media would lack the ability to control a computer to execute local transactions. This is an important failing of the combination of references because it is local transactions which specify how to compose different events into complex execution units or services.

Valuations (Functional Model)

The functional model defines how the occurrence of events of a class change the values of attributes of said class. This is done by means of valuations. A valuation describes how the occurrence of an event of a class changes the value of an attribute of said class. By defining at least a valuation for every attribute of a class upon the occurrence of an event of said class, the formal language specification enables the complete and precise definition of how the occurrence of said event affects the values of all the attributes of said class. This means that valuations describe the behaviour of classes in response to events. Valuations are not taught in any of the prior art references so the function model of which valuations are a part is an important claim limitation which is not found in the applied references.

Figure(s) in the prosecuted application and text in the parent and CIP specifications that support the functional model limitation in claims 1-3

Figure 5: from page 12

> "FIG. 5 illustrates an exemplary dialog for receiving input for the functional model."

Figure 15: from page 13

> FIG. 15 is a dialog box to enter the functional model formulas that define evaluation of the attribute "cause" with the "modify" event (an event is a single service). The functional model relates services mathematically through well-formed formulas to the values of attributes these services act upon.

[Col. 7, lines 29-32]

> "In one implementation, the Conceptual Model is subdivided into four complementary models: an object model, a dynamic model, a functional model, and a presentation model."

[Col. 8, lines 2-5]

> "Rather, four complementary models, that of the object model, the dynamic model, the functional model and the presentation model, are employed to allow a designer to specify the system requirements."

[Col. 8, lines 31-37]

> "In accordance with the widely accepted object oriented conceptual modeling principles, the Conceptual Model is subdivided into an object model, a dynamic model, and a functional model. These three models, however, are insufficient by themselves to specify a complete application, because a complete application also

requires a user interface."

[Col. 13, lines 59-67 to Col. 14, lines 1-15]

"One embodiment of the present invention, however, employs a functional model that is quite different with respect to these conventional approaches. In this functional model, the semantics associated with any change of an object state is captured as a consequence of an event occurrence. To do this, the following information is declaratively specified: how every event changes the object state depending on the arguments of the involved event, and the object,s current state. This is called. "valuation."

In particular, the functional model employs the concept of the categorization of valuations. Three types of valuations are defined.:push-pop, state-independent and discrete-domain based. Each type fixes the pattern of information required to define its functionality.

From page 26 of the clean substitute CIP specification (not found in parent):

"One embodiment of the present invention, however, employs a functional model that is quite different with respect to these conventional approaches. In this functional model, the semantics associated with any change of an object state is captured as a consequence of an event occurrence. Basically, the functional model allows a SOSY modeler to specify a class, an attribute of that class and an event of that class and then define a mathematical or logical formula that defines how the attribute's value will be changed when this event happens. An "event" as used in the claims means a single service and not a transaction which is defined as a composed or complex service (which means more than one service executes). In the preferred embodiment, condition-action pair is specified for each valuation. The condition is a single math or logic formula is specified which specifies a condition which results in a value or logical value which can be mapped to only one of two possible values: true or false. The action is a single math or logical formula which specifies how the value of the attribute is changed if the service is executed and the condition is true. In other embodiments, only a single formula that specifies the change to the attribute if the service is executed is required.

The functional model is built in the preferred embodiment by presenting a dialog box that allows the user to choose a class, an attribute of that class and a service of that class and then fill in one or more formula or logical expressions (condition-action or only action) which controls how the value of that attribute will be changed when the service is executed. The important thing about this is that the user be allowed to specify the mathematical or logical operation which will be performed to change the value of the attribute when the service is executed, and it is not critical how the user interface is implemented. Any means to allow a user to specify the class, the attribute of that class and the service of that class and then fill in a mathematical or logical expression which controls what happens to the specified attribute when the service is executed will suffice to practice the invention. Every one of these mathematical expressions is referred to as a valuation. Every valuation has to have a condition and action pair in the preferred embodiment, but in other species, only an action need be specified. The condition can be any well formed formula resulting in a Boolean value which can be mapped to only one of two possible conditions: true or false. The action specified in the pair is any other well-formed mathematical and/or logical formula resulting in a new value for the variable attribute, said new value being of the attribute's same data type (type of data of action must be compatible with the type of data of the attribute). This valuation formula can be only mathematical or only a Boolean logical expression or a combination of both mathematical operators and Boolean logical expressions.

Regardless of the user interface used to gather data from the user to define the valuations in the functional model, all species within the genus of the invention of generating functional models will generate a data structure having the following content: data defining the valuation formula which affects the value of each variable attribute (the data that defines the valuation formula identifies the service and the attribute affected and the mathematical and/or logical operations to be performed and any operands needed). This data structure can be any format, but it must contain at least the above identified content.

To define the functional model, the following information is declaratively specified by the SOSY modeler: how every event changes the object state depending on the arguments of the involved event, and the object's current state. This is called "valuation".

In particular, the functional model employs the concept of the categorization of valuations. Three types of valuations are defined.:push-pop, state-independent and discrete-domain based. Each type fixes the pattern of information required to define its functionality.

Push-pop valuations are those whose relevant events increase or decrease the value of the attribute by a given quantity, or reset the attribute to a certain value.

State-independent valuations give a new value to the attribute involved independently of the previous attribute's value.

Discrete-domain valuations give a value to the attributes from a limited domain based on the attribute's previous value. The different values of this domain model the valid situations that are possible for the attribute."

[Col. 14, lines 34-44]

"This categorization of the valuations is a contribution of one aspect of the present invention that allows a complete formal specification to be generated in an automated way, completely capturing a event's functionality.

Accordingly, the functional model is responsible for capturing the semantics of every change of state for the attributes of a class. It has no graphical diagram. Textual information is collected through an interactive dialog that fills the corresponding part of the Information Structures explained before. FIG. 5 illustrates an exemplary dialog for receiving input for the functional model."

[Col.18, lines 53-65]

"Evaluations are formulas of the form F [a] F ' whose semantics is given by defining a r function that, from a ground action a returns a function between possible worlds. In other words, being a possible world for an object any valid state, the r function determines which transitions between object states are valid after the execution of an action a. In the example,there are the following evaluations:

[loan( )] book_count=book_count+1;

[returns( )] book_count=book_count-1;

Within this dynamic logic environment, the formula F is evaluated in s Î W, and F ' is evaluated in r (a), with r (a) being the world represented by the object state after the execution in s of the action considered."

[Col. 20, lines 41-43]

"Finally, the functional model yields the dynamic formulas related to evaluations, where the effect of events on attributes is specified."

Distinction over prior art

This limitation (the concept of valuations) is not taught in any of the prior art references. The examiner cites that Goodwin teaches this in col. 17, lines 2-5 ("run the code generator 330 to generate the source code objects that will support the services in the Interface Definition Language (IDL) file that describes the interface of the objects to the business applications") but what this teaches is the generation of some source code that will implement the interface (defined in terms of the Interface Definition Language) of services, not the actual body or implementation (the behaviour) of said services.

If the Examiner has a passage which actually teaches a functional model with the concept of valuations, the applicants hereby request that such a passage be properly identified and requests the Examiner to explain how the passage teaches same.

Therefore, none of the references nor a combination of them could teach a process to automatically produce the complete source code of an application from a formal language specification, because said source code would lack the portion that performs the calculations associated to valuations (the behaviour of a class in response to the occurrence of an event).

Dynamic Model Limitation

Purpose

The dynamic model specifies the behaviour of an object in response to services, triggers and global transactions. The Dynamic Model encompasses two diagrams:

- The State Transition Diagram, which enables the specification of valid lives for an object (which services can occur to an object depending on which services have previously happened(
- The Object interaction Diagram, which enables the specification of:
    - o Trigger relationships, which determines which service will execute when a given condition defined for a given object holds.
    - o Global Transactions, which allows the composition of different services belonging to different objects can be composed into a more complex service.

Figure(s) in the prosecuted application

Figure 4A (State Transition Diagram)

Figure 4B (Object Interaction Diagram)

Figure 17 (State Transition Diagram)

Passages in the prosecuted application and the issued patent of the parent

case that provide evidence for interpretation of the Dynamic Model limitation

From Page 7 Of The Clean Substitute Specification:

"FIG. 17 is one of the two graphical user interface diagrams of the dynamic model on which the SOSY modeler has drawn a graphic illustrating the state transitions for the "expense" class."

[Col. 12, lines 15-44]

"The dynamic model specifies the behavior of an object in response to services, triggers and global transactions. In one embodiment, the dynamic model is represented by two diagrams, a state transition diagram and an object interaction diagram.
The state transition diagram (STD) is used to describe correct behavior by establishing valid object life cycles for every class. A valid life refers to an appropriate sequence of states that characterizes the correct behavior of the objects that belong to a specific class. Transitions represent valid changes of state. A transition has an action and, optionally, a control condition or guard. An action is composed of a service plus a subset of its valid agents defined in the Object Model. If all of them are marked, the transition is labeled with an asterisk (*). Control conditions are well formed formulas defined on object attributes and/or service arguments to avoid the possible non-determinism for a given action. Actions might have one precondition that must be satisfied in order to accept its execution. A blank circle represents the state previous to existence of the object. Transitions that have this state as source must be composed of creation actions. Similarly, a bull's eye represent the state after destruction of the object. Transitions having this state as destination must be composed of destruction actions. Intermediate states are represented by circles labeled with an state name.

Accordingly, the state transition diagram shows a graphical representation of the various states of an object and transitions between the states."

Distinction over prior art

None of the references teach the concept of a State Transition Diagram (which can be seen as a State Machine) to specify valid lives for an object. The examiner says that this is anticipated in Goodwin in col. 6, lines 64-67 to col.7, lines 1-7 ("system also provides a data server (described in reference to FIGS.3 and 7) for performing run-time object queries that are transformed to access information from enterprise resources with results instantiated between business objects ... generated within the composed object service framework... accomplished through the use of an object query service (OMG compliant) and provides a CORBA service to which clients can submit object queries over generated objects and instantiated objects into objects with the composed behaviors defined by the framework") The cited passage does not teach either the concept of State Transition Diagrams, nor the concept of State Machines, but the use of an object query service, which is in no way related to State Machines nor State Transition Diagrams. If the Examiner has a passage which actually teaches a dynamic model state transition diagram or object interaction diagram, the applicants hereby request that such a passage be properly identified and requests the Examiner to explain how the passage teaches a dynamic model.

The applicant could go on with further explanation of limitations that are in claims 1-3 which are not found in the applied prior art, but the above recited differences and knowledge missing from the prior art references are sufficient to obviate the obviousness rejection for lack of suggestion.

Where the prior art of a combination of references cited in support of an obviousness rejection does not teach an element needed to solve the problem the claimed invention solved, the obviousness argument must fail. In re Hayes Microcomputer Products, Inc., 982 F.2d 1527, 1541, 25 USPQ2d 1241 (Fed. Cir. 1992) [failure of prior art to teach a claimed method of detecting escape sequences in modems doomed obviousness invalidity argument of infringer even though escape sequences themselves were admittedly in the prior art]. There is no suggestion to support an obviousness rejection based upon a combination of references where the combination of references does not contain all the knowledge needed to make the claimed invention.

A reference may be said to teach away from a proposed combination in support of an obviousness rejection when a person of ordinary skill in the art, upon reading the reference, would be led in a direction divergent from the path that was taken by the applicant to solve the problem the claimed invention solved or would be discouraged from using the teachings of the reference in attempting to solve the problem the claimed invention solved. In re Gurley, 27 F.3d 551, 553, 31 USPQ2d 1130, 1131 (Fed. Cir. 1994). In general, a reference will teach away if it suggests that the line of development flowing from the reference's disclosure is unlikely to be productive of the result sought by the applicant. 27 F.3d at 533, 31 USPQ2d at 1131. References taken in combination teach away when they would produce a "seemingly inoperative device". 27 F.3d at 553, 31 USPQ2d at 1131-1132. In re Caldwell, 319 F.2d 254, 256, 138 USPQ 243, 245 (CCPA1963) (reference teaches away if it leaves the impression that the product would not have the property sought by the applicant).

Here, even if the references applied by the Examiner to reject claims 1-3 were combined, the combination would not have the properties of the claimed invention, *i.e.,* the ability to translate local transactions and events specified in the object model, valuations specified in the functional model and data about object lives specified in state transition diagrams and object interactions specified in the object interaction diagram of the dynamic model into working code.

Accordingly, the Examiner is respectfully requested to withdraw the obviousness rejection of claims 1-3.

New Claims

New claims have been added to claim the invention in different ways than claims 1-3 to cover the full breadth of the applicant's invention. New claim 4 is directed to specifying the process carried out to determine what primitives are in the formal specification the user defined and then to write one or more code modules that implement those primitives and interact with each other in the way specified by the execution model. Claim 4 differs from claim 1 in the sense that claim 4 requires a step of determining whether the formal specification defines the need for certain functions in the program to be written, and then, if so, writing one or more code modules which implements the one or more functions specified in the formal specification. These code modules are written so as to interact with the other code modules in accordance with the execution model.

The execution model is inherently specified in every formal language specification encoding a Conceptual Model. It is essentially hard wired, because every program needs to perform its functions in a logical order, and the execution model just specifies what that logical order is. What this means for example is that in every program, if there are preconditions specified in the formal language specification, those preconditions need to be evaluated first to determine if they are satisfied before any subsequent function such as calculation of a valuation formula can be performed. Likewise, if the formal language specification specifies that there are trigger conditions that will be true (cause some event such as invoking a particular service when the trigger condition becomes true) if the value of certain variables changes to some specified value or Boolean logic state, and the formal language specification specifies a valuation formula to alter the value of said variable, then the execution model requires that the valulation formula be calculated first and then the results must be evaluated to determine if the trigger condition is triggered. New claim 4 is intended to claim this process of determining what functions are required in the target program to be written as specified in the primitives defined by the user in the formal language specification and then write code modules that implement the functions defined by those primitives in the correct order specified by the execution model.

New claims 5-7 depend from new claim 4 and add various details to the broad statement of claim 4.

New claim 8 claims an apparatus programmed to perform the translation process to translate from the formal language specification to full and complete source code. The claim specifies the functions performed while implementing a process which follows the execution model to extract information from the formal language specification and convert it to code.

Dependent claim 9 defines the apparatus as programmed to provide user interface tools by which various models within the conceptual model can be composed.

Dependent claim 10 defines an apparatus programmed to present GUI tools to define the primitives of the conceptual model and to validate the resulting formal language specification.

New claims 11-13 define the translator invention in terms of computer-readable medium storing instructions which control a computer to carry out the translation, GUI front end and validation processing.

New claims 14-16 define the translation process in another way in terms of the three major components that will be in the automatically generated code.

New claim 17 defines the translation process in terms of the tiers in the finally generated code.

New claims 18-43 define a validation process which is a key precursor process to the translation process as no translation will be allowed until the validation process is complete and all errors have been located and corrected in the formal language specification.

Respectfully submitted,

Dated: May 26, 2005

[illegible]

Ronald Craig Fish
Reg. No. 28,843
Tel 408 778 3624
FAX 408 776 0426

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail, postage prepaid, in an envelope addressed to: Mail Stop RCE, Commissioner for Patents , P.O. Box 1450, Alexandria, Va. 22313-1450.
on 5/26/05
(Date of Deposit)

[illegible]

Ronald Craig Fish, President
Ronald Craig Fish, a Law Corporation
Reg. No. 28,843